**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ  
ГЛАВА 1. ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ «ОПТОВАЯ БАЗА ДАННЫХ»  
1.1. Описание предметной области  
1.2. Понятие БД и СУБД  
1.3. Типы моделей баз данных  
ГЛАВА 2. ПРОЕКТИРОВАНИЕ РЕЛЯЦИОННОЙ БАЗЫ ДАННЫХ «ОПТОВАЯ БАЗА ДАННЫХ»  
2.1. Логическая модель базы данных (ER-диаграмма)  
2.2. Описание таблиц и их структуры  
2.3. Связи между таблицами  
2.4. Описание используемых ограничений (Constraints)  
2.5. Вычисляемые столбцы (Computed Columns)  
ГЛАВА 3. ОРГАНИЗАЦИЯ ВЫБОРКИ ИНФОРМАЦИИ ИЗ БД (РАЗРАБОТКА ПРЕДСТАВЛЕНИЙ)  
3.1. Простые представления  
3.2. Сложные представления  
ГЛАВА 4. ПРОЕКТИРОВАНИЕ ПРОГРАММНЫХ ОБЪЕКТОВ БАЗЫ ДАННЫХ  
4.1. Пользовательские функции (User-Defined Functions - UDFs)  
4.2. Хранимые процедуры (Stored Procedures - SPs)  
4.3. Триггеры (Triggers)  
4.4. Транзакции  
ГЛАВА 5. ТРЕБОВАНИЯ К ТЕХНИЧЕСКОМУ ОБЕСПЕЧЕНИЮ  
ГЛАВА 6. ИНСТРУКЦИЯ ПО ЭКСПЛУАТАЦИИ БАЗЫ ДАННЫХ  
6.1. Установка и настройка  
6.2. Основные операции с базой данных  
ЗАКЛЮЧЕНИЕ  
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ  
ПРИЛОЖЕНИЕ А (Листинг SQL-скрипта создания базы данных)

ВВЕДЕНИЕ

В современной экономике эффективное управление данными является ключом к успешной деятельности любого предприятия. Оптовая торговля, характеризующаяся большими объемами товарных потоков, широким кругом контрагентов и сложными логистическими цепочками, особенно нуждается в автоматизации процессов учета и анализа информации. Разработка специализированной базы данных (БД) для оптовой компании позволяет систематизировать информацию о товарах, поставщиках, клиентах, заказах и складских остатках, что способствует повышению оперативности принятия управленческих решений, снижению вероятности ошибок, связанных с человеческим фактором, и улучшению качества обслуживания клиентов.

**Актуальность темы** курсовой работы обусловлена растущей потребностью оптовых предприятий в надежных и гибких инструментах для управления информацией. Внедрение хорошо спроектированной базы данных позволяет оптимизировать процессы закупок, продаж, складского учета, а также проводить анализ деятельности для выявления тенденций и повышения рентабельности.

**Целью курсовой работы** является разработка и реализация реляционной базы данных «Оптовая база данных», предназначенной для эффективного управления ключевыми процессами оптовой компании. Достижение этой цели включает получение и углубление знаний в области проектирования структуры баз данных, разработке и реализации их моделей, а также приобретение практических навыков работы с современной системой управления базами данных (СУБД) Microsoft SQL Server.

**Для достижения поставленной цели были определены следующие задачи:**

1. Проанализировать и формализовать предметную область «Оптовая база данных», выделив основные сущности и их атрибуты.
2. Спроектировать логическую структуру базы данных, включая определение таблиц, их полей, типов данных и первичных ключей. Рекомендуется использовать ER-диаграмму для наглядного представления структуры.
3. Реализовать спроектированную базу данных средствами СУБД Microsoft SQL Server (указать используемую версию, например, SQL Server 2019 Developer Edition).
4. Определить и реализовать взаимосвязи между таблицами с помощью внешних ключей, а также другие необходимые ограничения целостности данных (UNIQUE, CHECK, DEFAULT).
5. Разработать набор представлений (VIEWs) для упрощения типовых запросов и предоставления данных пользователям в удобном формате.
6. Разработать пользовательские функции (UDFs), хранимые процедуры (SPs) и триггеры для инкапсуляции бизнес-логики, автоматизации часто выполняемых операций и обеспечения целостности данных при их модификации.
7. Изучить и применить механизм транзакций для обеспечения атомарности и согласованности операций, изменяющих данные в нескольких таблицах.

**Объектом исследования** являются процессы сбора, хранения, обработки и анализа информации, характерные для деятельности оптовой компании.  
**Предметом исследования** выступает модель реляционной базы данных «Оптовая база данных» и средства ее реализации и эксплуатации в среде СУБД Microsoft SQL Server.

**Практическая значимость** работы заключается в том, что разработанная база данных может служить прототипом или основой для создания полнофункциональной информационной системы оптового предприятия. Предложенные решения по структуре данных, представлениям, процедурам и триггерам направлены на повышение эффективности учета и управления товарно-материальными ценностями, взаимоотношениями с клиентами и поставщиками.

ГЛАВА 1. ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ «ОПТОВАЯ БАЗА ДАННЫХ»

**1.1. Описание предметной области**

Предметная область «Оптовая база данных» охватывает основные аспекты деятельности компании, занимающейся оптовыми продажами. Для эффективного управления такой компанией необходимо учитывать множество сущностей и процессов. В рамках данной курсовой работы были выделены следующие ключевые сущности:

1. **Категории товаров (Categories):** Группировка товаров по общим признакам (например, "Электроника", "Продукты питания").
2. **Поставщики (Suppliers):** Организации или лица, поставляющие товары.
3. **Товары (Products):** Непосредственно реализуемая продукция с указанием цен закупки и продажи, единиц измерения и связи с категориями и поставщиками.
4. **Склады (Warehouses):** Места хранения товаров.
5. **Остатки на складе (WarehouseStock):** Информация о количестве каждого товара на конкретном складе, включая минимальные и максимальные уровни запасов.
6. **Должности сотрудников (Positions):** Перечень должностей в компании.
7. **Сотрудники (Employees):** Персонал компании, ответственный за различные операции.
8. **Клиенты/Заказчики (Customers):** Физические или юридические лица, приобретающие товары.
9. **Заказы (Orders):** Запросы от клиентов на поставку товаров.
10. **Детали заказа (OrderDetails):** Конкретные товарные позиции в рамках одного заказа с указанием количества, цены и скидки.
11. **Способы оплаты (PaymentMethods):** Возможные варианты оплаты заказов.
12. **Платежи (Payments):** Факты оплаты заказов.
13. **Журнал аудита сотрудников (EmployeeAuditLog):** Таблица для отслеживания изменений ключевой информации о сотрудниках.

**Основные бизнес-процессы, которые призвана автоматизировать разрабатываемая база данных:**

1. **Управление каталогом продукции:** Добавление, изменение, удаление информации о товарах и их категориях.
2. **Управление базой поставщиков:** Ведение информации о поставщиках, их контактных данных.
3. **Учет движения товаров на складах:** Регистрация поступления товаров, отгрузки по заказам, отслеживание текущих остатков, контроль за достижением минимальных/максимальных уровней запасов.
4. **Кадровый учет:** Ведение информации о сотрудниках, их должностях, датах приема на работу.
5. **Управление взаимоотношениями с клиентами (CRM):** Хранение информации о клиентах, их контактных данных, истории заказов, предоставление скидок.
6. **Обработка заказов клиентов:** Создание новых заказов, добавление товарных позиций, расчет итоговой суммы с учетом скидок, отслеживание статуса выполнения заказа (от "Ожидает обработки" до "Доставлен" или "Отменен").
7. **Финансовые операции:** Регистрация платежей от клиентов, привязка платежей к заказам.
8. **Аудит и безопасность:** Отслеживание изменений важных данных, таких как информация о сотрудниках.

**1.2. Понятие БД и СУБД**

**База данных (БД)** — это представленная в объективной форме совокупность самостоятельных материалов (статей, расчетов, нормативных актов, судебных решений и иных подобных материалов), систематизированных таким образом, чтобы эти материалы могли быть найдены и обработаны с помощью электронной вычислительной машины (ЭВМ). В более прикладном смысле, БД — это организованная структура, предназначенная для хранения, изменения и обработки взаимосвязанной информации, преимущественно больших объёмов.

**Система управления базами данных (СУБД)** — это комплекс программных и языковых средств, необходимых для создания баз данных, поддержания их в актуальном состоянии и организации поиска в них необходимой информации. СУБД обеспечивает:

1. Управление данными во внешней памяти (на дисках).
2. Управление данными в оперативной памяти.
3. Журнализацию изменений, резервное копирование и восстановление БД после сбоев.
4. Поддержку языков БД (язык определения данных, язык манипулирования данными).
5. Обеспечение безопасности и целостности данных.
6. Управление транзакциями.
7. Поддержку многопользовательского режима работы.

В данной курсовой работе для реализации базы данных «Оптовая база данных» используется реляционная СУБД **Microsoft SQL Server 2019 Developer Edition**.

**1.3. Типы моделей баз данных**

Существует несколько основных моделей данных, используемых при проектировании баз данных:

1. **Иерархическая модель:** Данные организованы в виде древовидной структуры. Каждый узел (запись) может иметь только одного "родителя", но несколько "потомков". Доступ к данным осуществляется по иерархическим связям сверху вниз.
2. **Сетевая модель:** Является расширением иерархической. В этой модели запись может иметь несколько "родителей" и несколько "потомков", что позволяет моделировать более сложные связи типа "многие-ко-многим".
3. **Реляционная модель:** Данные представляются в виде набора двумерных таблиц (отношений). Каждая таблица состоит из строк (кортежей) и столбцов (атрибутов). Связи между таблицами устанавливаются посредством общих атрибутов (ключей). Эта модель получила наибольшее распространение благодаря своей простоте, гибкости, строгому математическому аппарату и поддержке мощного языка запросов SQL.

Для проекта «Оптовая база данных» была выбрана **реляционная модель**, поскольку она наилучшим образом подходит для структурированного хранения данных об оптовой торговле, позволяет легко устанавливать связи между различными сущностями (товары, клиенты, заказы и т.д.), обеспечивает гибкость при формировании запросов и хорошо поддерживается современными СУБД, включая MS SQL Server.

ГЛАВА 2. ПРОЕКТИРОВАНИЕ РЕЛЯЦИОННОЙ БАЗЫ ДАННЫХ «ОПТОВАЯ БАЗА ДАННЫХ»

**2.1. Логическая модель базы данных (ER-диаграмма)**

Логическая модель базы данных представляет собой абстрактное представление данных и связей между ними, не зависящее от конкретной СУБД. Одним из наиболее распространенных способов визуализации логической модели является ER-диаграмма (Entity-Relationship Diagram).

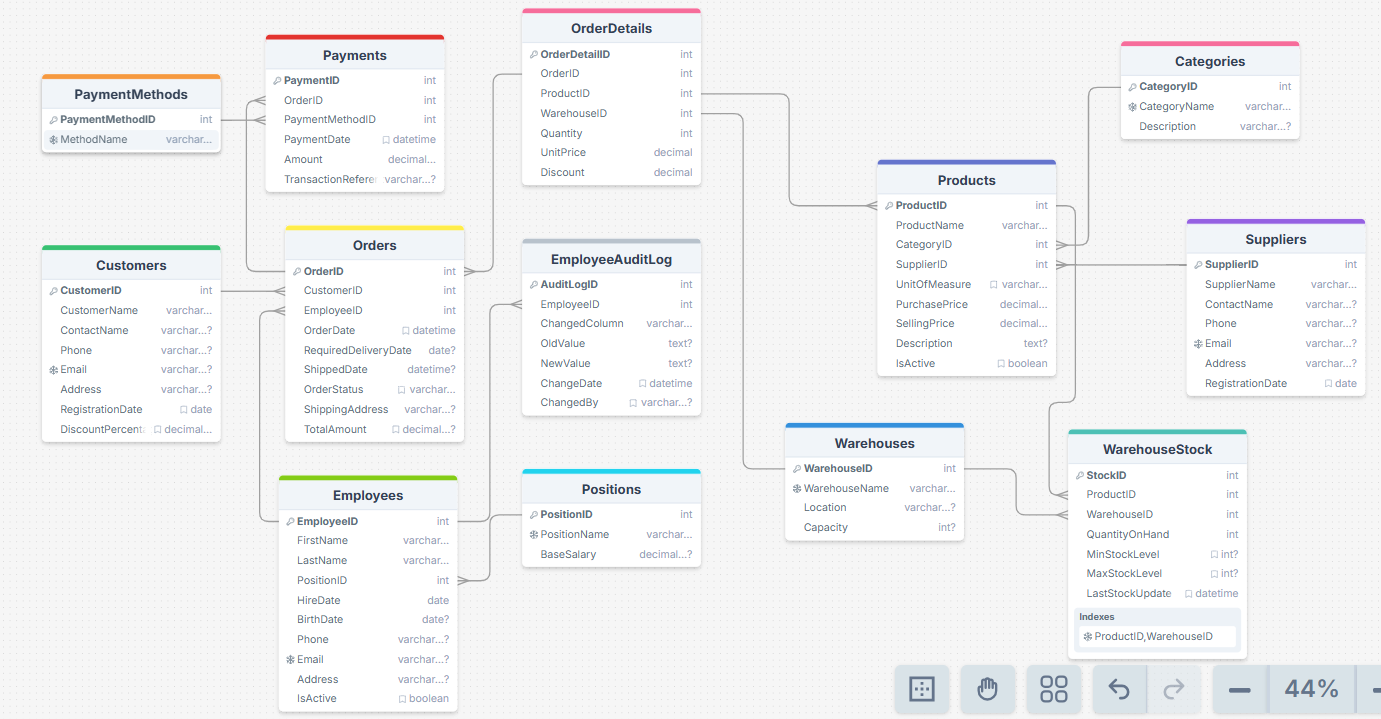


Рисунок 2.1.1 – ER диаграмма базы данных «Оптовая база»

**2.2. Описание таблиц и их структуры**

Ниже представлено описание каждой таблицы, созданной в базе данных WholesaleDB, с указанием назначения и структуры полей.

**Таблица Categories (Категории товаров)**

Назначение: Хранение информации о категориях товаров.

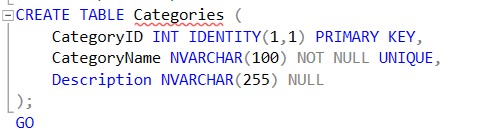


Рисунок 2.2.1 – SQL код создания таблицы Categories

**Таблица Suppliers (Поставщики)**

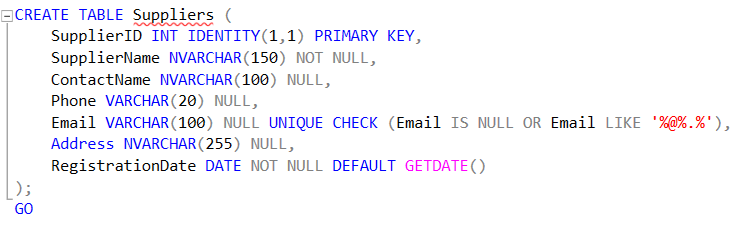
Назначение: Хранение информации о поставщиках товаров.  


Рисунок 2.2.2 – SQL код создания таблицы Suppliers

**Таблица Products (Товары)**

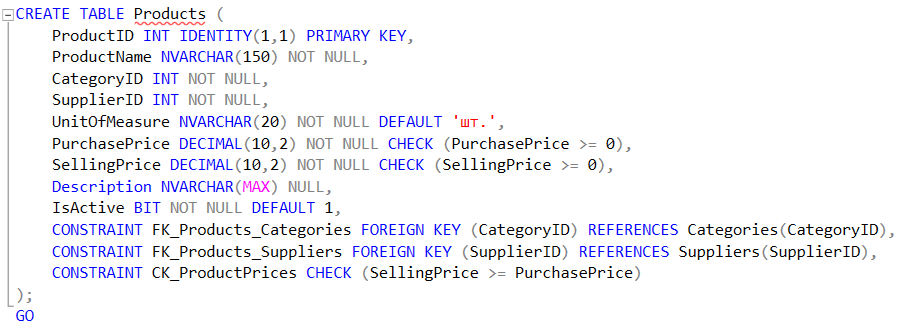
Назначение: Хранение детальной информации о товарах.  


Рисунок 2.2.3 – SQL код создания таблицы Products

**Таблица Warehouses (Склады)**

Назначение: Хранение информации о складах компании.

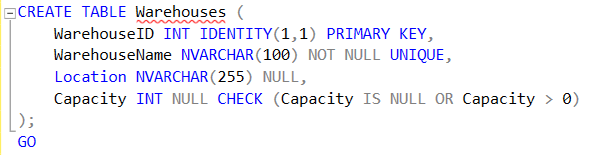


Рисунок 2.2.4 – SQL код создания таблицы Warehouses

**Таблица WarehouseStock (Остатки на складе)**

Назначение: Учет количества каждого товара на конкретном складе.

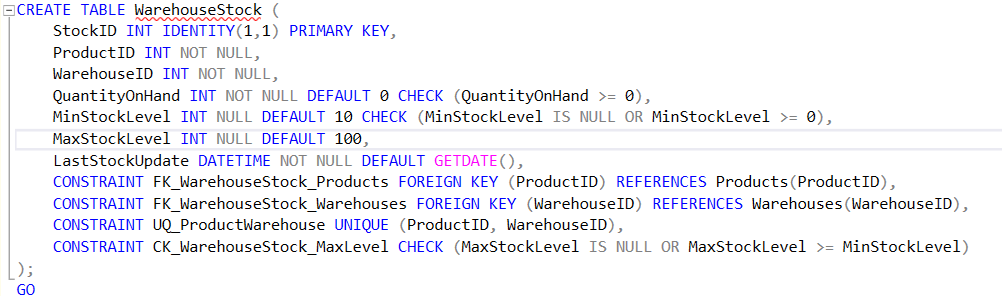


Рисунок 2.2.5 – SQL код создания таблицы WarehouseStock

**Таблица Positions (Должности сотрудников)**

Назначение: Справочник должностей сотрудников.

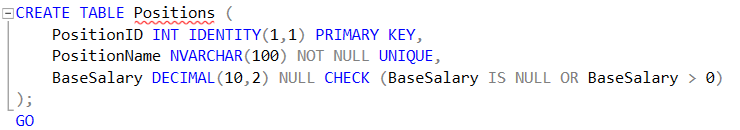


Рисунок 2.2.6 – SQL код создания таблицы Positions

**Таблица Employees (Сотрудники)**

Назначение: Хранение информации о сотрудниках компании.

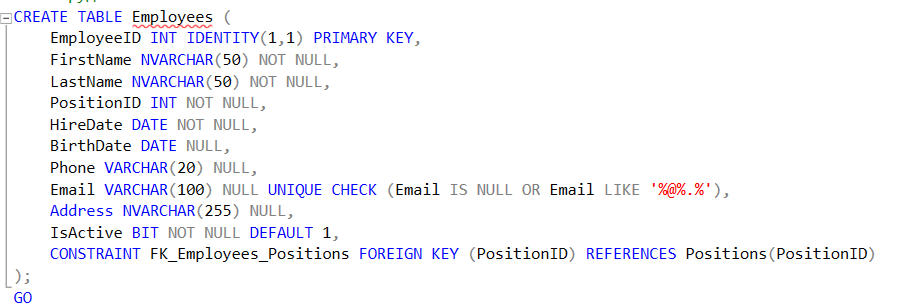


Рисунок 2.2.7 – SQL код создания таблицы Employess

**Таблица Customers (Клиенты/Заказчики)**

Назначение: Хранение информации о клиентах компании.

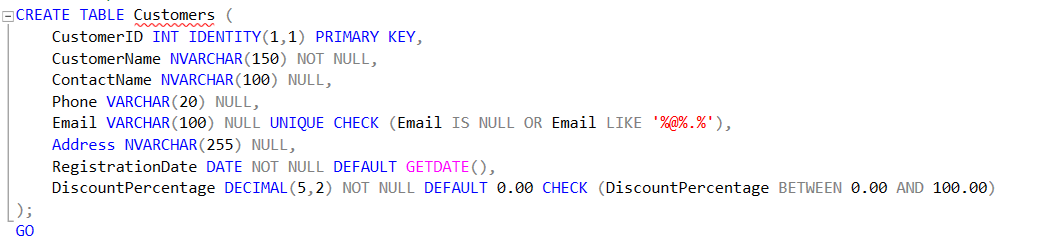


Рисунок 2.2.8 – SQL код создания таблицы Customers

**Таблица Orders (Заказы)**

Назначение: Хранение информации о заказах клиентов.

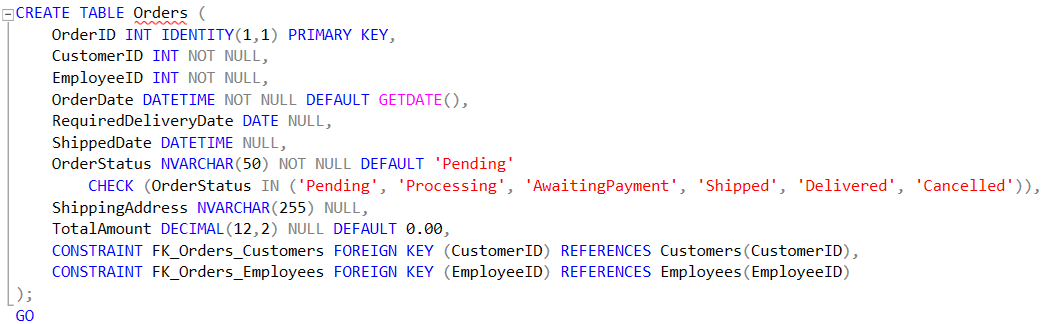


Рисунок 2.2.9 – SQL код создания таблицы Orders

**Таблица OrderDetails (Детали заказа)**

Назначение: Хранение товарных позиций каждого заказа.

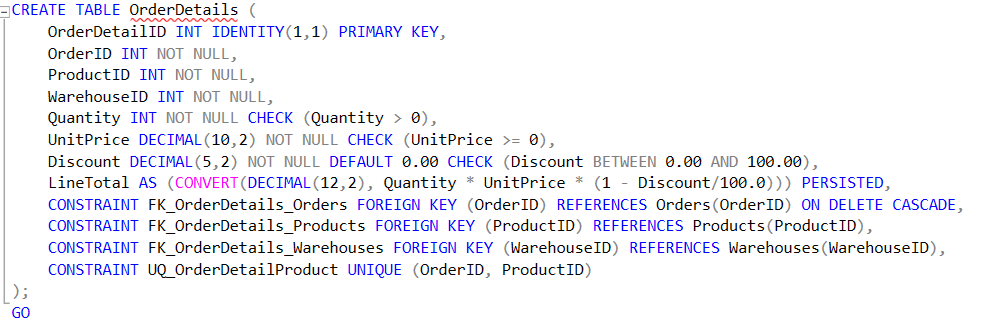
Структура:  


Рисунок 2.2.10 – SQL код создания таблицы OrderDetail

**Таблица PaymentMethods (Способы оплаты)**

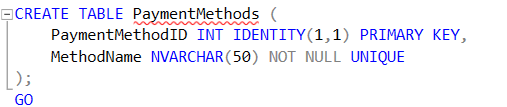
Назначение: Справочник доступных способов оплаты.  


Рисунок 2.2.11 – SQL код создания таблицы PaymentMethods

**Таблица Payments (Платежи)**

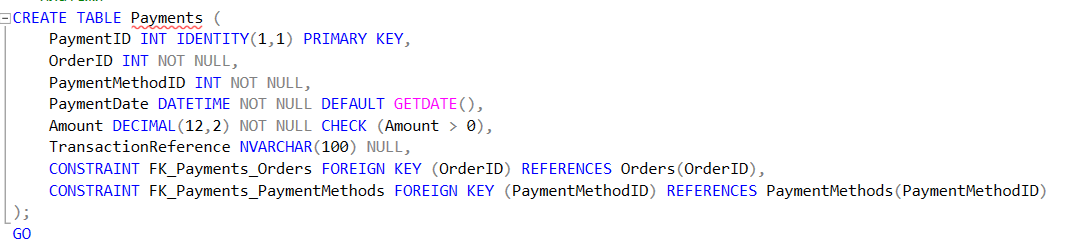
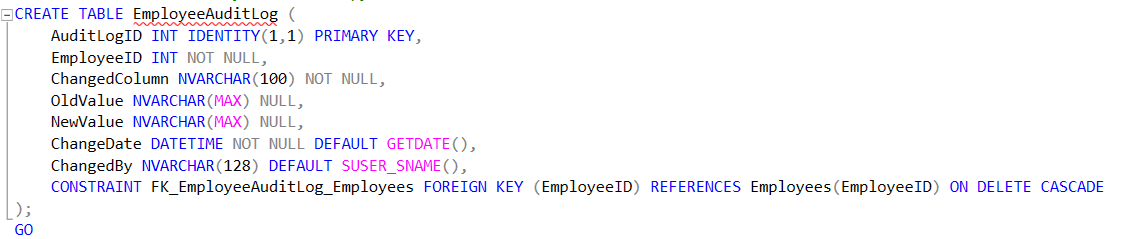
**Назначение**: Хранение информации о платежах по заказам.  


Рисунок 2.2.12 – SQL код создания таблицы Payments

**Таблица EmployeeAuditLog (Журнал аудита сотрудников)**

**Назначение**: Запись изменений ключевых данных сотрудников для аудита.  
 Рисунок 2.2.13 – SQL код создания таблицы EmployeeAuditLog

**2.3. Связи между таблицами**

Связи между таблицами в реляционной базе данных устанавливаются с помощью первичных (**Primary Key** 🡪 **PK**) и внешних (**Foreign Key** 🡪 **FK**) ключей. В нашей базе данных реализованы следующие основные связи типа "один-ко-многим" (1:M):

* **Categories** (1) 🡪 **Products** (M): Одна категория может включать множество товаров, но каждый товар принадлежит только одной категории. Связь обеспечивается полем CategoryID.
* **Suppliers** (1) 🡪 **Products** (M): Один поставщик может поставлять множество товаров, но каждый товар (в рамках данной модели) поставляется одним основным поставщиком. Связь по полю SupplierID.
* **Warehouses** (1) 🡪 **WarehouseStock** (M): На одном складе могут храниться остатки множества различных товаров. Связь по полю WarehouseID.
* **Products** (1) 🡪 **WarehouseStock** (M): Один товар может иметь остатки на нескольких складах. Связь по полю ProductID.
* **Positions** (1) 🡪 **Employees** (M): Одну должность могут занимать несколько сотрудников, но каждый сотрудник имеет одну основную должность. Связь по полю PositionID.
* **Customers** (1) 🡪 **Orders** (M): Один клиент может сделать множество заказов. Связь по полю CustomerID.
* **Employees** (1) 🡪 **Orders** (M): Один сотрудник может оформить множество заказов. Связь по полю EmployeeID.
* **Orders** (1) 🡪 **OrderDetails** (M): Один заказ может содержать множество товарных позиций. Связь по полю OrderID. **Важно:** для этой связи установлено правило ON DELETE CASCADE, что означает, что при удалении заказа будут автоматически удалены все связанные с ним детали заказа.
* **Products** (1) 🡪 **OrderDetails** (M): Один товар может присутствовать во множестве деталей различных заказов. Связь по полю ProductID.
* **Warehouses** (1) 🡪 **OrderDetails** (M): С одного склада могут отгружаться товары для множества деталей различных заказов. Связь по полю WarehouseID.
* **PaymentMethods** (1) 🡪 **Payments** (M): Один способ оплаты может использоваться для множества платежей. Связь по полю PaymentMethodID.
* **Orders** (1) 🡪 **Payments** (M): К одному заказу может быть привязано несколько платежей (например, частичная оплата). Связь по полю OrderID.
* **Employees** (1) 🡪 **EmployeeAuditLog** (M): Для одного сотрудника может быть множество записей в журнале аудита. Связь по полю EmployeeID. **Важно:** для этой связи установлено правило ON DELETE CASCADE, что означает, что при удалении сотрудника будут автоматически удалены все связанные с ним записи аудита.

**2.4. Описание используемых ограничений (Constraints)**

Ограничения целостности данных играют важную роль в обеспечении корректности и согласованности информации в базе данных.

* **PRIMARY KEY (PK):** Гарантирует уникальность каждой строки в таблице. Каждая таблица в WholesaleDB имеет первичный ключ, обычно это поле IDENTITY (например, ProductID в Products, OrderID в Orders).
* **FOREIGN KEY (FK):** Обеспечивает ссылочную целостность между таблицами. Например, Products.CategoryID ссылается на Categories.CategoryID. Если для внешнего ключа указано ON DELETE CASCADE (как в OrderDetails.OrderID и EmployeeAuditLog.EmployeeID), то при удалении родительской записи (например, заказа) автоматически удаляются все связанные дочерние записи (детали этого заказа). Это помогает поддерживать согласованность данных.
* **UNIQUE:** Гарантирует уникальность значения в столбце или наборе столбцов, не являющихся первичным ключом. Примеры: Categories.CategoryName, Suppliers.Email, Warehouses.WarehouseName, UQ\_ProductWarehouse (комбинация ProductID и WarehouseID в WarehouseStock), UQ\_OrderDetailProduct (комбинация OrderID и ProductID в OrderDetails).

**CHECK:** Позволяет определить условие, которому должны удовлетворять данные в столбце. Примеры:

* Email **LIKE '%@%.%'** в таблицах Suppliers, Employees, Customers для базовой проверки формата email.
* **PurchasePrice >= 0 и SellingPrice >= 0** в Products.
* **CK\_ProductPrices CHECK (SellingPrice >= PurchasePrice)** в Products гарантирует, что цена продажи не ниже цены закупки.
* **QuantityOnHand >= 0** в WarehouseStock.
* **CK\_WarehouseStock\_MaxLevel CHECK (MaxStockLevel IS NULL OR MaxStockLevel >= MinStockLevel)** в WarehouseStock проверяет корректность максимального уровня запасов.
* **DiscountPercentage BETWEEN 0.00 AND 100.00** в Customers и OrderDetails.
* **OrderStatus IN ('Pending', 'Processing', 'AwaitingPayment', 'Shipped', 'Delivered', 'Cancelled'**) в Orders ограничивает возможные статусы заказа.

**DEFAULT:** Задает значение по умолчанию для столбца, если оно не указано при вставке новой строки. Примеры: GETDATE() для RegistrationDate в Suppliers и Customers, 'шт.' для UnitOfMeasure в Products, 1 для IsActive в Products и Employees, 'Pending' для OrderStatus в Orders.

**2.5. Вычисляемые столбцы (Computed Columns)**

Вычисляемый столбец — это виртуальный столбец, который физически не хранится в таблице, если он не помечен как PERSISTED. Его значение вычисляется на основе других столбцов в той же таблице.

В таблице **OrderDetails** используется вычисляемый столбец **LineTotal:  
LineTotal AS (CONVERT(DECIMAL(12,2), Quantity \* UnitPrice \* (1 - Discount/100.0))) PERSISTED**

* **Назначение:** Автоматически рассчитывает итоговую стоимость товарной позиции с учетом количества, цены за единицу и скидки.
* **Формула:** Quantity \* UnitPrice \* (1 - Discount/100.0). CONVERT(DECIMAL(12,2), ...) используется для явного приведения результата к нужному типу данных с двумя знаками после запятой.
* **PERSISTED:** Это ключевое слово означает, что значения вычисляемого столбца физически сохраняются в таблице. Это позволяет создавать индексы на таком столбце и может улучшить производительность запросов, фильтрующих или сортирующих по этому столбцу, за счет избежания повторных вычислений. Однако это немного увеличивает накладные расходы при операциях INSERT и UPDATE.

ГЛАВА 3. ОРГАНИЗАЦИЯ ВЫБОРКИ ИНФОРМАЦИИ ИЗ БД (РАЗРАБОТКА ПРЕДСТАВЛЕНИЙ)

Представления (**VIEW**) — это сохраненные SQL-запросы, которые ведут себя как виртуальные таблицы. Они не хранят данные сами по себе (за исключением индексированных представлений), а извлекают их из одной или нескольких базовых таблиц в момент обращения. Представления используются для:

1. Упрощения сложных запросов.
2. Предоставления пользователям только необходимых данных (безопасность на уровне столбцов и строк).
3. Обеспечения логической независимости данных (структура базовых таблиц может меняться, а представление останется прежним, если изменения не затрагивают его логику).

**Разработаны следующие представления:**

* 1. **Простые представления**

**v\_CustomerContactInfo (Контактная информация клиентов)**

**Назначение**: Предоставляет основную контактную информацию о клиентах.

**SQL-код:**

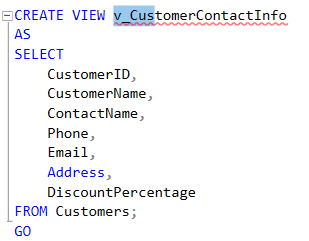
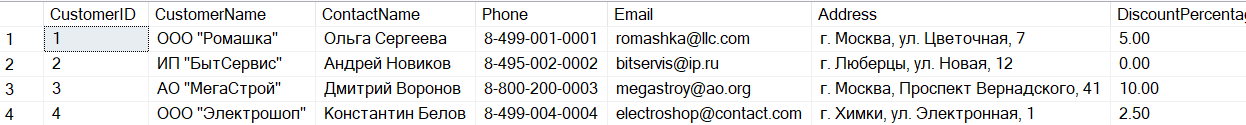


Рисунок 3.1.1 – SQL код создания представления CustomerContactInfo

**Результат выполнения: **Рисунок 3.1.2 – SQL код выполнения представления CustomerContactInfo

**v\_EmployeeDirectory (Справочник сотрудников)**

**Назначение**: Отображает основную информацию о сотрудниках, включая их должность.

**SQL-код:**

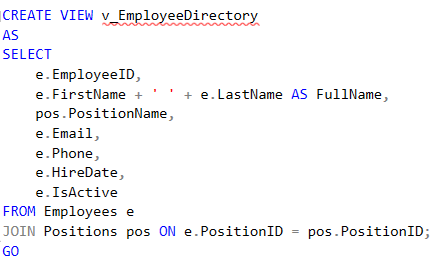
****

Рисунок 3.1.3 – SQL код создания представления EmployeeDirectory

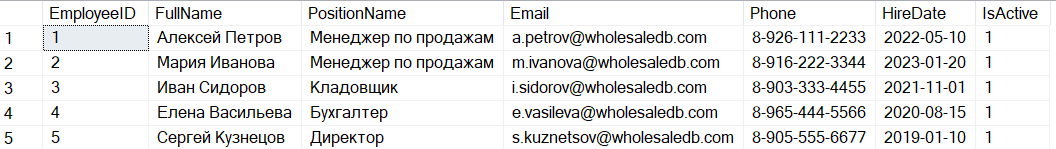
**Результат выполнения**

Рисунок 3.1.4 – SQL код выполнения представления EmployeeDirectory

**3.2. Сложные представления**

**v\_ActiveProducts (Активные товары)**

**Назначение**: Отображает информацию об активных товарах, включая их категорию, поставщика, цену продажи и общий текущий остаток на всех складах (с использованием пользовательской функции fn\_GetProductCurrentStock).

**Код SQL:**

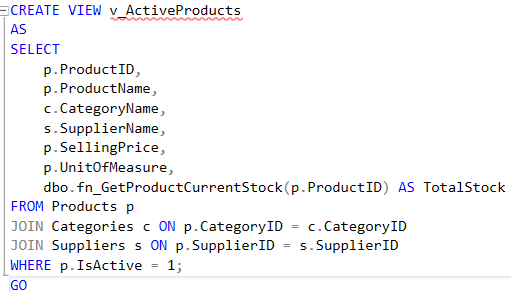


Рисунок 3.2.5 – SQL код создания представления ActiveProducts

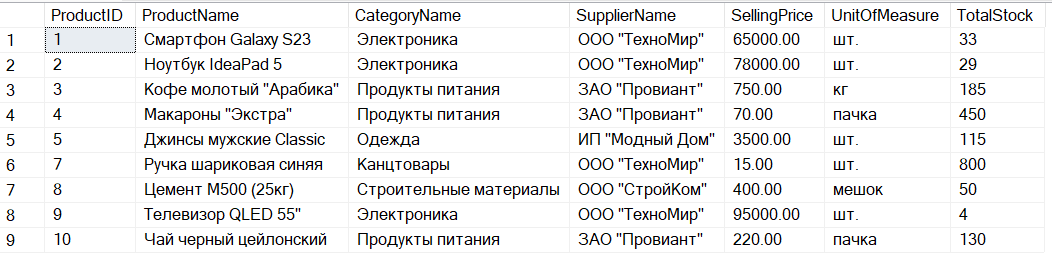
**Результат выполнения:  
**

Рисунок 3.2.6 – SQL код выполнения представления ActiveProducts

**v\_PendingOrders (Заказы в обработке)**

**Назначение**: Показывает заказы, которые находятся в статусах 'Pending' (Ожидает), 'Processing' (В обработке) или 'AwaitingPayment' (Ожидает оплаты), с информацией о клиенте и ответственном сотруднике.

**SQL-code**

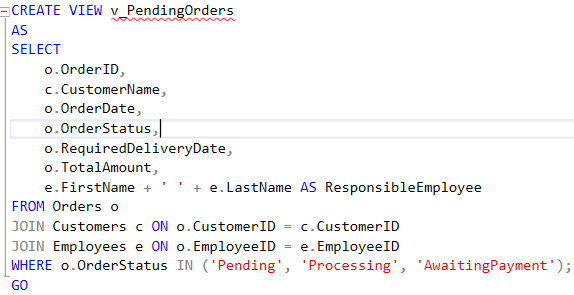


Рисунок 3.2.7 – SQL код создания представления PendingOrders

**Результат выполнения:**

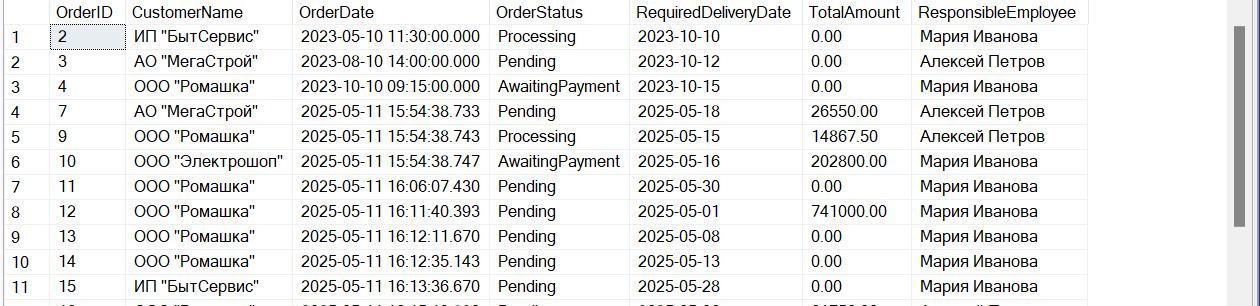


Рисунок 3.2.8 – SQL код выполнения представления PendingOrders

**v\_ProductStockSummary (Сводка по остаткам товаров)**

**Назначение**: Предоставляет сводную информацию об общем количестве каждого товара на всех складах.

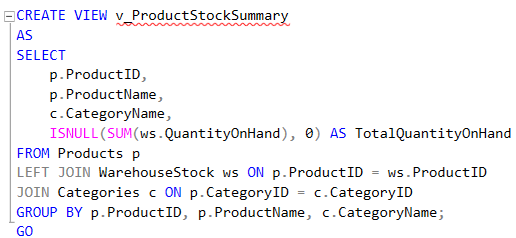
**SQL-код**: ****

Рисунок 3.2.9 – SQL код создания представления ProductStockSummary

**Результат выполнения**:

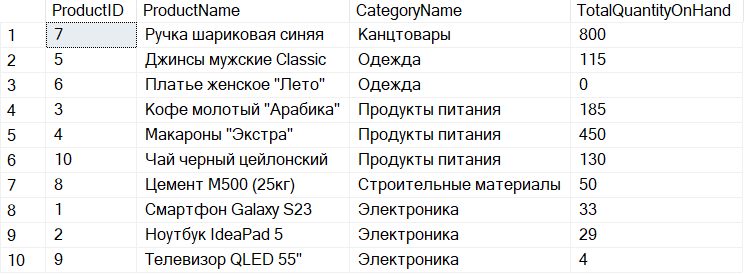
****

Рисунок 3.2.10 – SQL код выполнения представления ProductStockSummary

**v\_DetailedOrderInformation (Детальная информация о заказах)**

**Назначение**: Отображает расширенную информацию о заказах, включая данные клиента, сотрудника и детали доставки.

**SQL-код**:

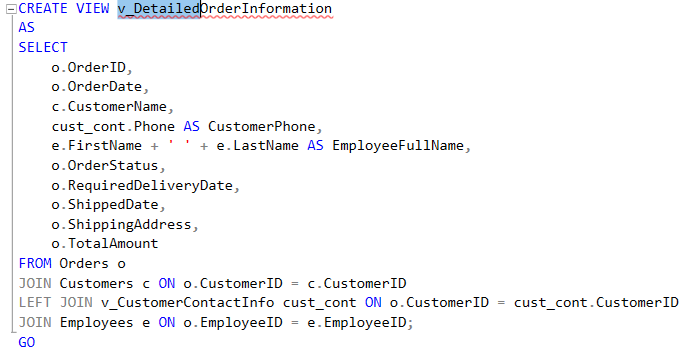


Рисунок 3.2.11 – SQL код создания представления DetailedOrderInformation

**Результат выполнения:**

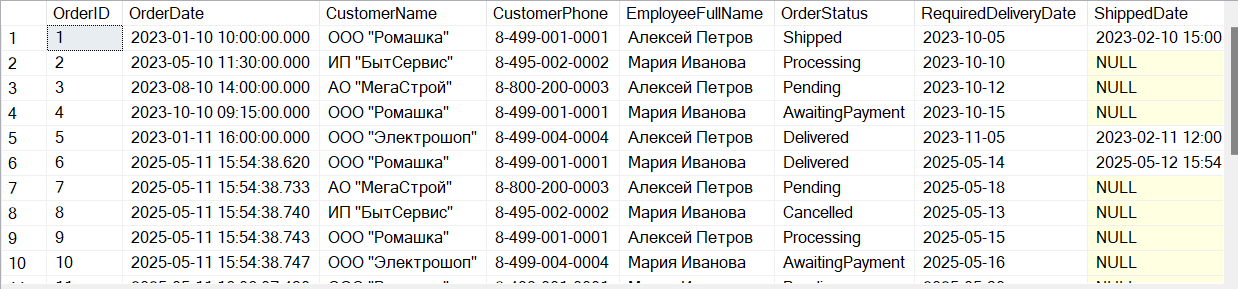
****

Рисунок 3.2.12 – SQL код выполнения представления DetailedOrderInformation

**v\_ProductSalesPerformance (Эффективность продаж товаров)**

**Назначение**: Показывает общее количество проданных единиц каждого товара и общую выручку от их продажи.

**SQL-код**:

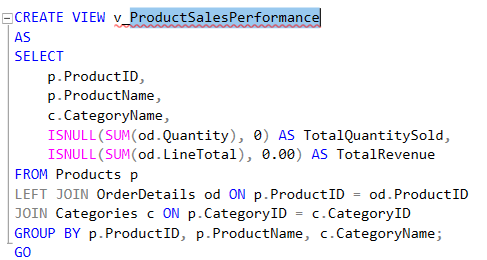


Рисунок 3.2.13 – SQL код создания представления ProductSalesPerformance

**Результат выполнения:**

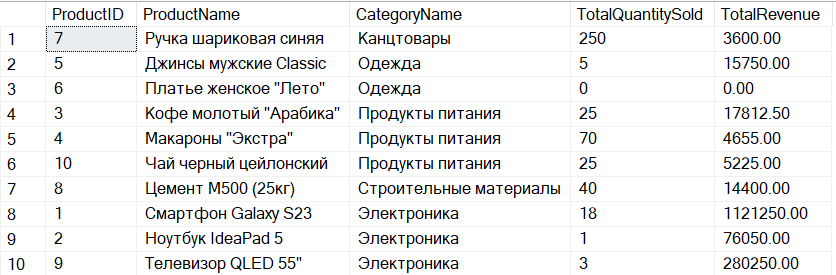
****

Рисунок 3.2.14 – SQL код выполнения представления ProductSalesPerformance

**v\_SupplierProductList (Список товаров по поставщикам)**

**Назначение**: Предоставляет список товаров, сгруппированных по поставщикам, с указанием цен и статуса активности.

**SQL-код**:

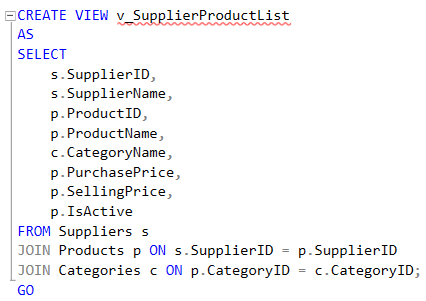


Рисунок 3.2.15 – SQL код создания представления SupplierProductList

**Результат выполнения:**

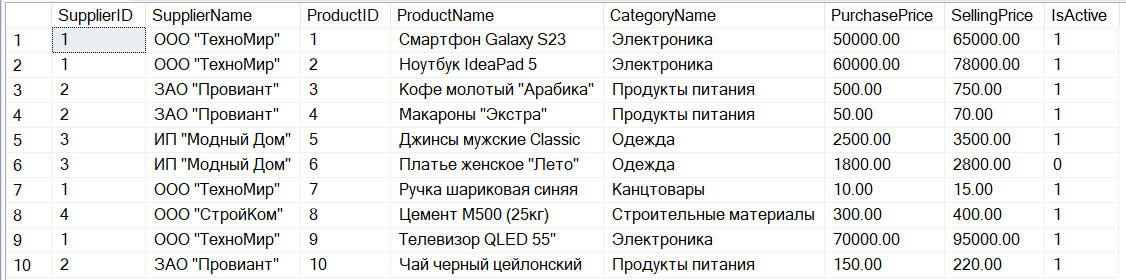


Рисунок 3.2.16 – SQL код выполнения представления SupplierProductList

ГЛАВА 4. ПРОЕКТИРОВАНИЕ ПРОГРАММНЫХ ОБЪЕКТОВ БАЗЫ ДАННЫХ

Программные объекты базы данных, такие как пользовательские функции, хранимые процедуры и триггеры, позволяют инкапсулировать бизнес-логику на стороне сервера, повысить производительность и обеспечить целостность данных.

**4.1. Пользовательские функции (User-Defined Functions - UDFs)**

Пользовательские функции расширяют возможности SQL, позволяя создавать собственные именованные блоки кода, которые могут принимать параметры и возвращать скалярное значение или таблицу.

**Скалярная функция fn\_GetProductCurrentStock**

**Назначение**: Возвращает общее текущее количество указанного товара на всех складах.

**Параметры**: @ProductID INT - идентификатор товара.

**Возвращаемое значение**: INT - общее количество товара.

**SQL-код:**

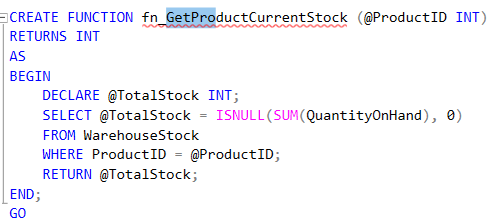
****

Рисунок 4.1.1 – SQL код создания функции GetProductCurrentStock

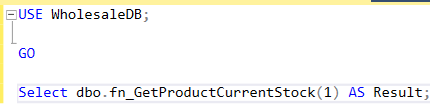
****

Рисунок 4.1.2 – Код для выполнения функции GetProductCurrentStock

**Результат выполнения:**

****

Рисунок 4.1.3 – результат выполнения функции GetProductCurrentStock

**Скалярная функция fn\_CalculateOrderTotalAmount**

**Назначение**: Рассчитывает и возвращает общую сумму указанного заказа на основе позиций в OrderDetails.

**Параметры**: @OrderID INT - идентификатор заказа.

**Возвращаемое значение**: DECIMAL(12,2) - общая сумма заказа.

**SQL-код:**

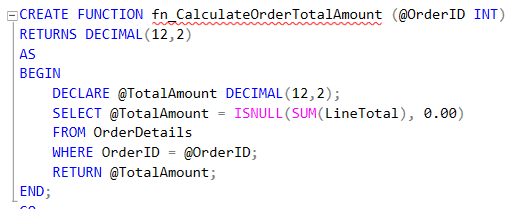
****

Рисунок 4.1.4 – SQL код создания функции CalculateOrderTotalAmount

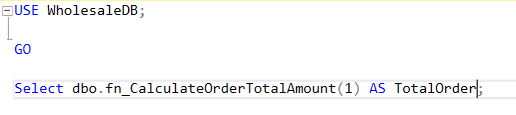


Рисунок 4.1.5 – Код для выполнения функции CalculateOrderTotalAmount

**Результат выполнения:**



Рисунок 4.1.6 – результат выполнения функции CalculateOrderTotalAmount

**Табличная функция fn\_GetCustomerOrderHistory**

**Назначение**: Возвращает историю заказов для указанного клиента, включая дату, статус, сумму и ФИО ответственного сотрудника.

**Параметры**: @CustomerID INT - идентификатор клиента.

**Возвращаемое значение**: Таблица с полями OrderID, OrderDate, OrderStatus, TotalAmount, EmployeeFullName.

**SQL-код:**

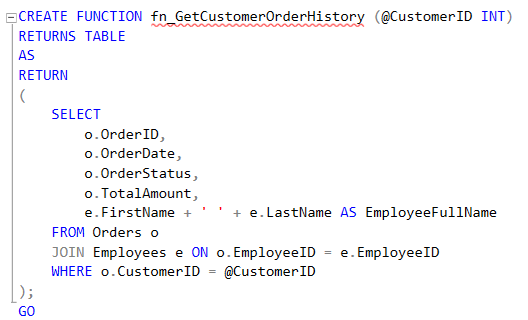
****

Рисунок 4.1.7 – SQL код создания функции GetCustomerOrderHistory

**Результат выполнения:**

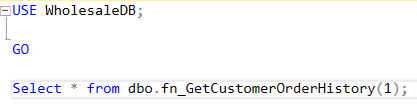


Рисунок 4.1.8 – Код для выполнения функции GetCustomerOrderHistory

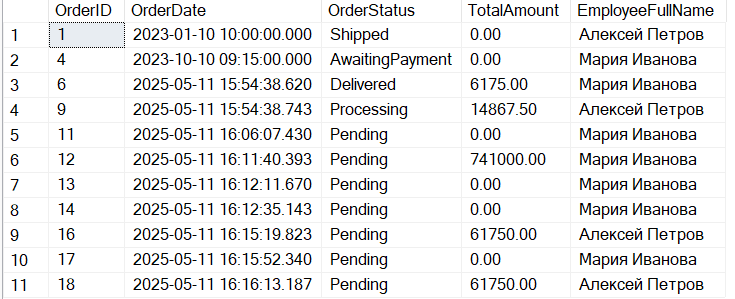


Рисунок 4.1.9 – результат выполнения функции GetCustomerOrderHistory

**4.2. Хранимые процедуры (Stored Procedures - SPs)**

Хранимые процедуры — это предварительно скомпилированные наборы SQL-инструкций, хранящиеся в базе данных. Они используются для выполнения часто повторяющихся задач, инкапсуляции сложной бизнес-логики, повышения производительности и безопасности.

**sp\_CreateNewOrder (Создание нового заказа)**

**Назначение**: Создает новую запись о заказе.

**Параметры**:

@CustomerID INT: Идентификатор клиента.

@EmployeeID INT: Идентификатор сотрудника, оформляющего заказ.

@RequiredDeliveryDate DATE = NULL (опционально): Требуемая дата доставки.

@ShippingAddress NVARCHAR(255) = NULL (опционально): Адрес доставки (если отличается от адреса клиента).

**Логика**: Проверяет существование клиента и активность сотрудника. Если адрес доставки не указан, используется адрес клиента по умолчанию. Создает новый заказ со статусом 'Pending' и нулевой общей суммой (сумма будет обновлена триггером при добавлении товаров). Возвращает ID созданного заказа.

**SQL-код:**

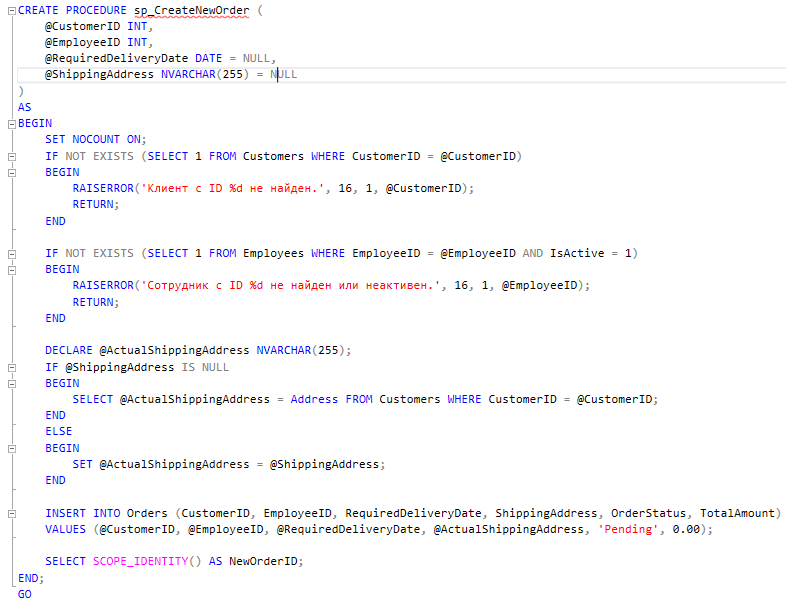


Рисунок 4.2.1 – SQL код создания процедуры CreateNewOrder

**Результат выполнения:**

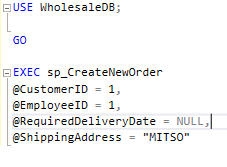
****

Рисунок 4.2.2 – Код для выполнения процедуры CreateNewOrder

****

Рисунок 4.2.3 – Результат выполнения процедуры CreateNewOrder

**sp\_AddProductToOrder (Добавление товара в заказ)**

**Назначение**: Добавляет товарную позицию в существующий заказ или обновляет количество, если товар уже есть в заказе.

Параметры:

@OrderID INT: Идентификатор заказа.

@ProductID INT: Идентификатор товара.

@WarehouseID INT: Идентификатор склада, с которого будет отгружен товар.

@Quantity INT: Количество товара.

**Логика**: Использует транзакцию. Проверяет существование заказа и его статус (можно редактировать только 'Pending', 'AwaitingPayment', 'Processing'), активность товара, существование склада. Проверяет наличие достаточного количества товара на складе. Если товар уже есть в заказе, увеличивает его количество; иначе добавляет новую позицию. Цена и скидка берутся из таблицы товаров и профиля клиента соответственно.

**SQL-код:**



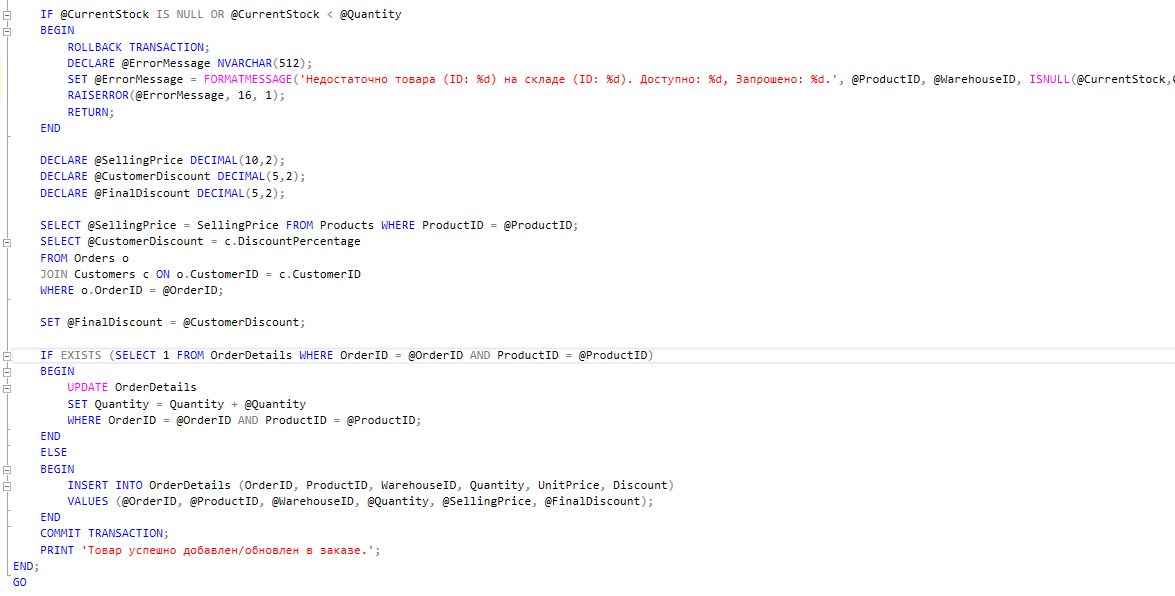


Рисунок 4.2.4 – SQL код создания процедуры AddProductToOrder

**Результат выполнения:**

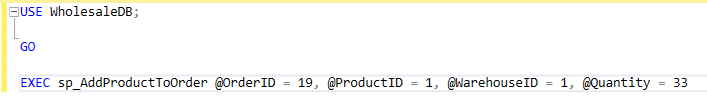


Рисунок 4.2.5 – Код для выполнения процедуры AddProductToOrder

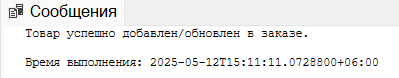


Рисунок 4.2.6 – Результат выполнения процедуры AddProductToOrder

**sp\_UpdateOrderStatus (Обновление статуса заказа)**

**Назначение**: Изменяет статус указанного заказа.

**Параметры**:

@OrderID INT: Идентификатор заказа.

@NewOrderStatus NVARCHAR(50): Новый статус заказа.

**Логика**: Проверяет существование заказа и корректность нового статуса. Если новый статус 'Shipped' и дата отгрузки еще не установлена, устанавливает текущую дату.

**SQL-код:**

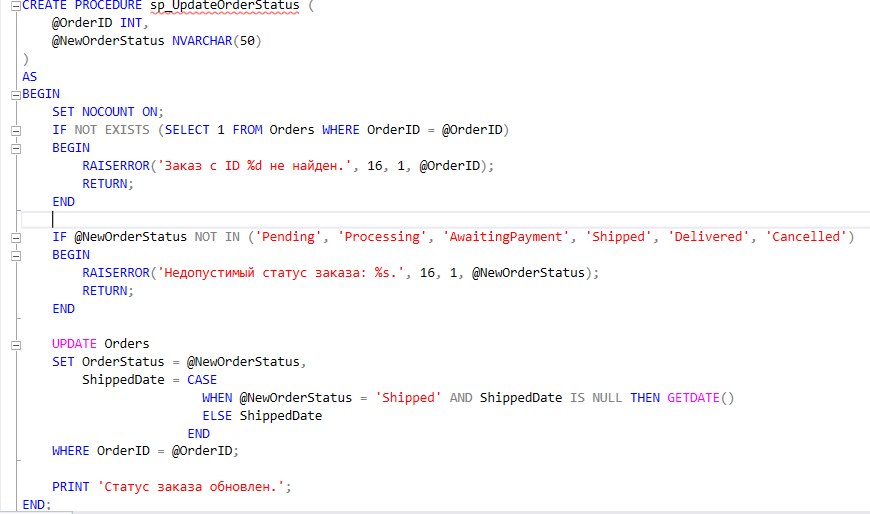


Рисунок 4.2.7 – SQL код создания процедуры UpdateOrderStatus

**sp\_ProcessPaymentForOrder (Обработка платежа по заказу)**

**Назначение**: Регистрирует платеж по заказу.

**Параметры**:

@OrderID INT: Идентификатор заказа.

@PaymentMethodID INT: Идентификатор способа оплаты.

@AmountPaid DECIMAL(12,2): Сумма платежа.

@TransactionReference NVARCHAR(100) = NULL (опционально): Ссылка на транзакцию.

**Логика**: Использует транзакцию. Проверяет существование заказа, способа оплаты и корректность суммы. Добавляет запись в таблицу Payments. Если заказ был в статусе 'AwaitingPayment' и общая сумма оплачена, изменяет статус заказа на 'Processing'.

**SQL-код**:

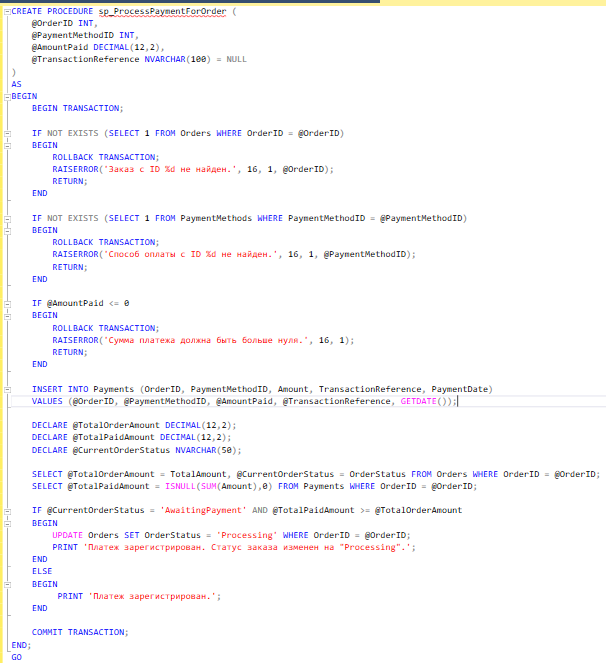


Рисунок 4.2.8 – SQL код создания процедуры ProcessPaymentForOrder

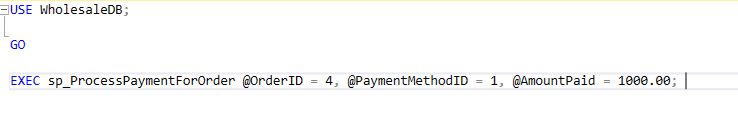


Рисунок 4.2.9 – Код для выполнения процедуры ProcessPaymentForOrder

**Результат выполнения:**

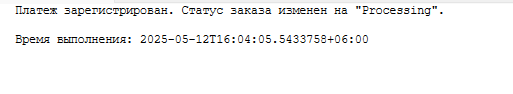
****

Рисунок 4.2.10 – Результат выполнения процедуры ProcessPaymentForOrder

**sp\_GetLowStockProducts (Получение товаров с низким остатком)**

**Назначение**: Выводит отчет о товарах, остаток которых на складе ниже или равен заданному проценту от минимального уровня запаса.

**Параметры**:

@TargetWarehouseID INT = NULL (опционально): Идентификатор конкретного склада (если NULL, то по всем складам).

@AlertThresholdPercentage DECIMAL(5,2) = 100.00 (опционально): Процент от минимального уровня. Если 100%, то товары, остаток которых равен или ниже минимального уровня.

**Логика**: Использует курсор для перебора остатков товаров на складах. Для каждого товара вычисляет пороговое значение (MinStockLevel \* @AlertThresholdPercentage / 100.0). Если текущий остаток меньше или равен этому порогу, информация о товаре выводится в консоль.

**SQL-код**:



Рисунок 4.2.10 – SQL код создания процедуры GetLowStockProducts

**Результат выполнения:**

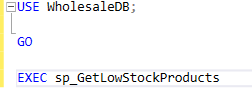
****

Рисунок 4.2.11 – Код для выполнения процедуры GetLowStockProducts

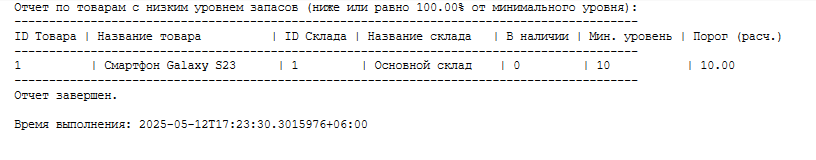
****

Рисунок 4.2.12 – Результат выполнения процедуры GetLowStockProducts

**4.3. Триггеры (Triggers)**

**Триггеры** — это специальный тип хранимых процедур, которые автоматически выполняются (срабатывают) в ответ на определенные события изменения данных (INSERT, UPDATE, DELETE) в указанной таблице. Они используются для поддержания целостности данных, каскадных обновлений/удалений, аудита.

**trg\_UpdateStockOnOrderDetailInsertDelete (Обновление остатков при изменении деталей заказа)**

**Событие**: AFTER INSERT, DELETE, UPDATE на таблице OrderDetails.

**Назначение**: Автоматически корректирует количество товаров на складе (WarehouseStock.QuantityOnHand) при добавлении, удалении или изменении количества/склада в позициях заказа. При добавлении товара в заказ — уменьшает остаток на складе; при удалении — увеличивает. При изменении — корректно обрабатывает старые и новые значения.

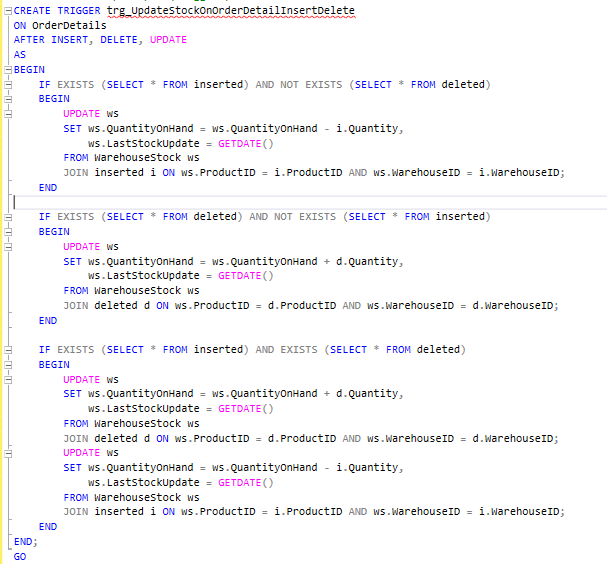
**SQL-код:**   


Рисунок 4.3.1 – SQL код создания процедуры UpdateStockOnOrderDetailInsertDelete

**trg\_UpdateOrderTotalAmount (Обновление общей суммы заказа)**

**Событие**: AFTER INSERT, UPDATE, DELETE на таблице OrderDetails.

**Назначение**: Автоматически пересчитывает и обновляет поле TotalAmount в таблице Orders при любых изменениях в связанных позициях заказа. Для расчета используется функция dbo.fn\_CalculateOrderTotalAmount.

**SQL-код:**



Рисунок 4.3.2 – SQL код создания процедуры UpdateOrderTotalAmount

**trg\_LogEmployeeChanges (Логирование изменений данных сотрудников)**

**Событие**: AFTER UPDATE на таблице Employees.

**Назначение**: Записывает информацию об изменениях в столбцах PositionID, IsActive, Email таблицы Employees в таблицу аудита EmployeeAuditLog. Фиксируются старое и новое значения, имя измененного столбца, дата изменения и пользователь, внесший изменения.

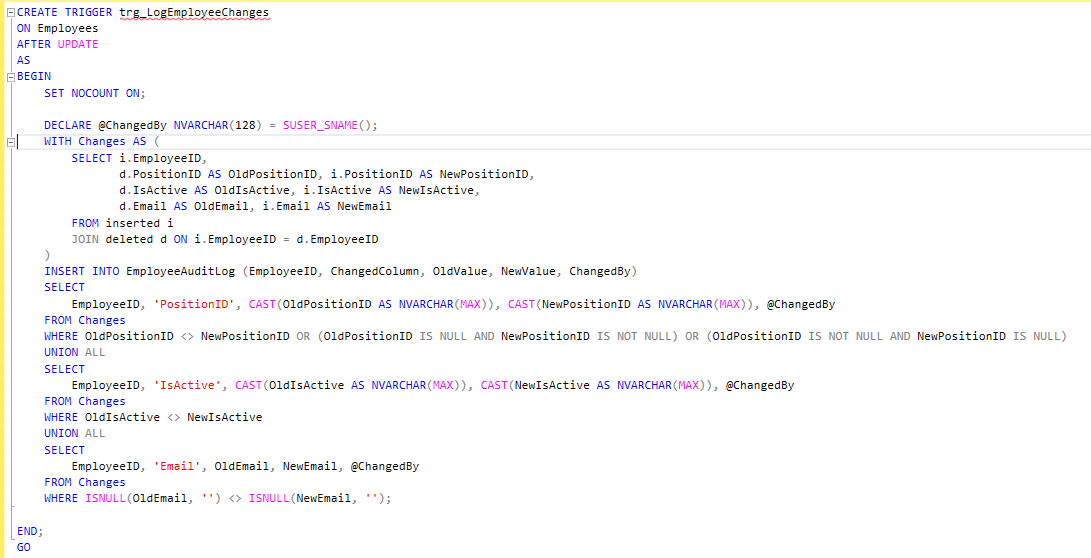
**SQL-код:** ****

Рисунок 4.3.3 – SQL код создания процедуры LogEmployeeChanges

**4.4. Транзакции**

**Транзакция** — это последовательность операций с базой данных, которая рассматривается как единое целое (атомарная операция). Либо все операции в транзакции успешно выполняются, либо ни одна из них не выполняется (в случае ошибки происходит откат). Транзакции обеспечивают свойства ACID:

**Atomicity (Атомарность):** Транзакция выполняется целиком или не выполняется вовсе.

**Consistency (Согласованность):** Транзакция переводит базу данных из одного согласованного состояния в другое.

**Isolation (Изолированность):** Параллельно выполняющиеся транзакции не должны влиять друг на друга.

**Durability (Долговечность):** Если транзакция успешно завершена, ее результаты сохраняются даже в случае сбоев системы.

Команды управления транзакциями в MS SQL Server:

**BEGIN TRANSACTION**: Начало транзакции.

**COMMIT TRANSACTION**: Успешное завершение транзакции, все изменения фиксируются.

**ROLLBACK TRANSACTION**: Откат транзакции, все изменения, сделанные с начала транзакции, отменяются.

В данной базе данных транзакции явно используются в следующих хранимых процедурах:

**sp\_AddProductToOrder**

**SQL-код:**

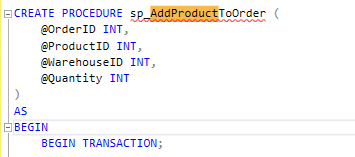


Рисунок 4.4.1 – Код начала первой транзакции в процедуре AddProductToOrder

Транзакция гарантирует, что если, например, товара не хватает на складе, то запись в OrderDetails не будет добавлена или изменена, предотвращая несогласованность данных.

**sp\_ProcessPaymentForOrder**:

**SQL-код:**

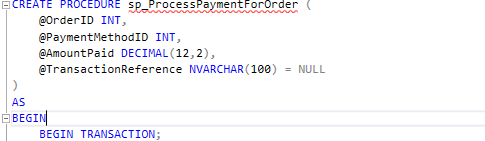


Рисунок 4.4.2 – Код начала второй транзакции в процедуре ProcessPaymentForOrder

Транзакция обеспечивает, что запись о платеже и возможное обновление статуса заказа произойдут вместе, либо ни одно из этих действий не будет выполнено в случае ошибки.

ГЛАВА 5. ТРЕБОВАНИЯ К ТЕХНИЧЕСКОМУ ОБЕСПЕЧЕНИЮ

Для успешной эксплуатации базы данных «Оптовая база данных» необходимо следующее техническое и программное обеспечение:

SQL Server Management Studio (SSMS) — это интегрированная среда для управления любой инфраструктурой SQL, от SQL Server до Azure SQL Database. SSMS предоставляет инструменты для настройки, мониторинга и администрирования экземпляров SQL Server и баз данных. Используйте SSMS для развёртывания, мониторинга и обновления компонентов уровня данных, используемых вашими приложениями, а также для создания запросов и скриптов.

Используйте SSMS для создания запросов, проектирования и управления базами данных и хранилищами данных, где бы они ни находились — на локальном компьютере или в облаке.

Для клиентов, которым требуется кроссплатформенное дополнение к SSMS для управления SQL и другими базами данных Azure, используйте Azure Data Studio.

SSMS 20.2.1 — последняя общедоступная (GA) версия.

Текущая версия SSMS поддерживает следующие 64-разрядные платформы при использовании с последним доступным пакетом обновлений:

**Поддерживаемые операционные системы:**

* Windows 11 (x64);
* Windows 10 (x64) версии 1607 (10.0.14393) и более поздние версии;
* Windows Server 2022 (x64);
* Windows Server Core 2022 (x64);
* Windows Server 2019 (x64);
* Windows Server Core 2019 (x64);
* Windows Server 2016 (x64) с .NET Framework 4.7.2.

**Дополнительные требования:**

* Для установки или обновления SSMS требуются права администратора.
* Для работы в среде виртуальной машины требуется полноценная операционная система Windows.
* Контейнеры Windows не поддерживаются.
* SSMS не поддерживается в решениях виртуализации приложений, таких как Microsoft App-V или MSIX для Windows, а также в сторонних технологиях виртуализации приложений.

Примечание. Чтобы установить SSMS на Windows Server Core, необходимо установить функцию совместимости приложений Server Core по запросу.

**Поддерживаемое оборудование:**

* Процессор x86 (Intel, AMD) 1,8 ГГц или быстрее. Рекомендуется двухъядерный или лучше.
* 2 ГБ ОЗУ; рекомендуется 4 ГБ ОЗУ (минимум 2,5 ГБ при работе на виртуальной машине).
* Место на жёстком диске: минимум 3 ГБ, до 10 ГБ свободного места.

Примечание. SSMS доступен только как 32-разрядное приложение для Windows. Если вам нужен инструмент, работающий на операционных системах, отличных от Windows, мы рекомендуем Azure Data Studio.

Поддерживаемые предложения SQL:

* Эта версия SSMS работает с SQL Server 2014 (12.x) и более поздними версиями. Она обеспечивает самую значительную поддержку для работы с новейшими облачными функциями в Azure SQL Database, Azure Synapse Analytics и Microsoft Fabric.
* Кроме того, SSMS 20.x можно установить вместе с SSMS 19.x, SSMS 18.x, SSMS 17.x и SSMS 16.x.
* Для служб SQL Server Integration Services (SSIS) SSMS 17.x и более поздние версии не поддерживают подключение к устаревшей службе SQL Server Integration Services. Чтобы подключиться к более ранней версии устаревших служб Integration Services, используйте версию SSMS, соответствующую версии SQL Server. Например, используйте SSMS 16.x для подключения к устаревшей службе SQL Server 2016 (13.x) Integration Services. SSMS 17.x и SSMS 16.x можно установить на одном компьютере. С момента выпуска SQL Server 2012 (11.x) база данных каталога SSIS, SSISDB, является рекомендуемым способом хранения, управления, запуска и мониторинга пакетов служб Integration Services [3].

ГЛАВА 6. ИНСТРУКЦИЯ ПО ЭКСПЛУАТАЦИИ БАЗЫ ДАННЫХ

**6.1. Установка и настройка**

Для развертывания базы данных «Оптовая база данных» необходимо выполнить следующие шаги:

1. Убедитесь, что на целевом сервере установлен Microsoft SQL Server (например, SQL Server 2019 Developer Edition) и на машине администратора/разработчика установлен SQL Server Management Studio (SSMS).
2. Запустите SSMS и подключитесь к вашему экземпляру SQL Server с правами, достаточными для создания баз данных и объектов.
3. В обозревателе объектов SSMS щелкните правой кнопкой мыши по узлу "Базы данных" и выберите "Создать базу данных...". Введите имя WholesaleDB и нажмите "ОК" (этот шаг можно пропустить, так как скрипт содержит команду CREATE DATABASE IF NOT EXISTS).
4. Откройте новое окно запроса (Ctrl+N или кнопка "Создать запрос").
5. Скопируйте весь SQL-код из файла WholesaleDBBase.sql (см. Приложение А) в это окно запроса.
6. Выполните скрипт, нажав F5 или кнопку "Выполнить". Скрипт создаст базу данных (если она не существует), все таблицы, представления, функции, хранимые процедуры, триггеры и заполнит таблицы начальными данными.
7. После успешного выполнения скрипта обновите список баз данных в обозревателе объектов. База данных WholesaleDB со всеми объектами должна появиться в списке.

**Работа с приложением Интерфейс БД WholesaleDB**

1.Зайти в папку site с помощью командной строки.  
2.В командной строке запустить сервер приложения с помощью node server.js

3.Открыть файл с проектом index.html

**Интерфейс приложения**:

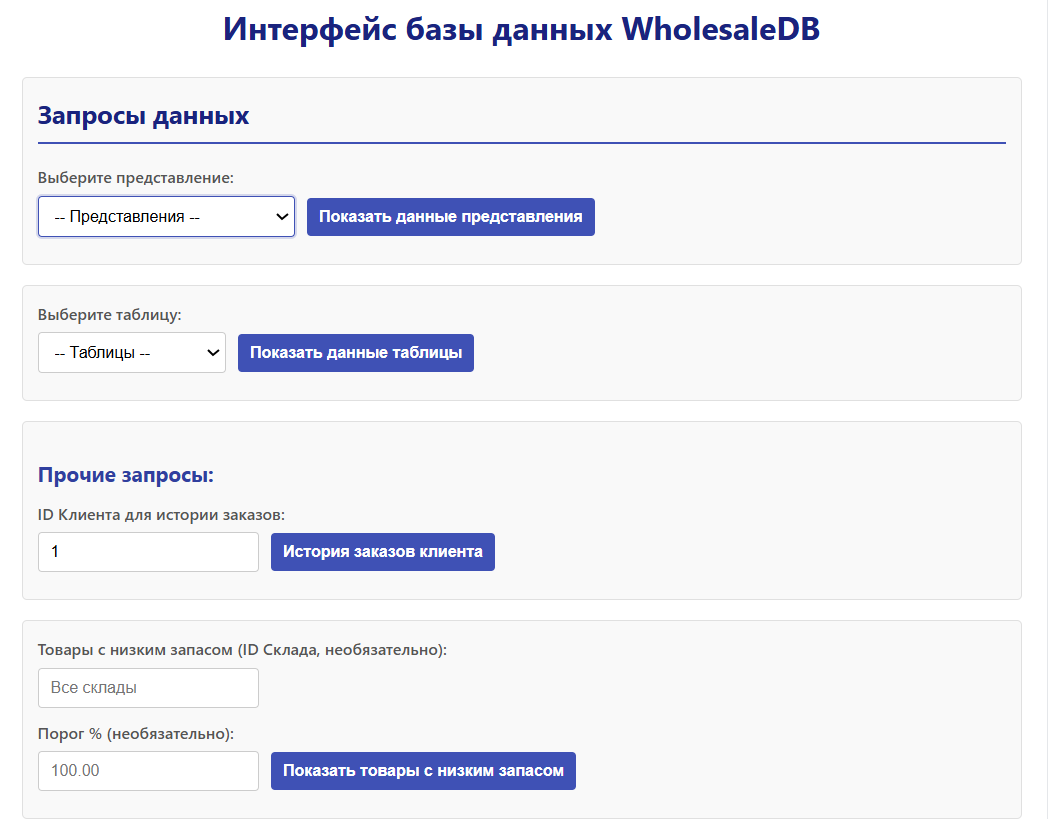


Рисунок 6.1 Интерфейс приложения(1)

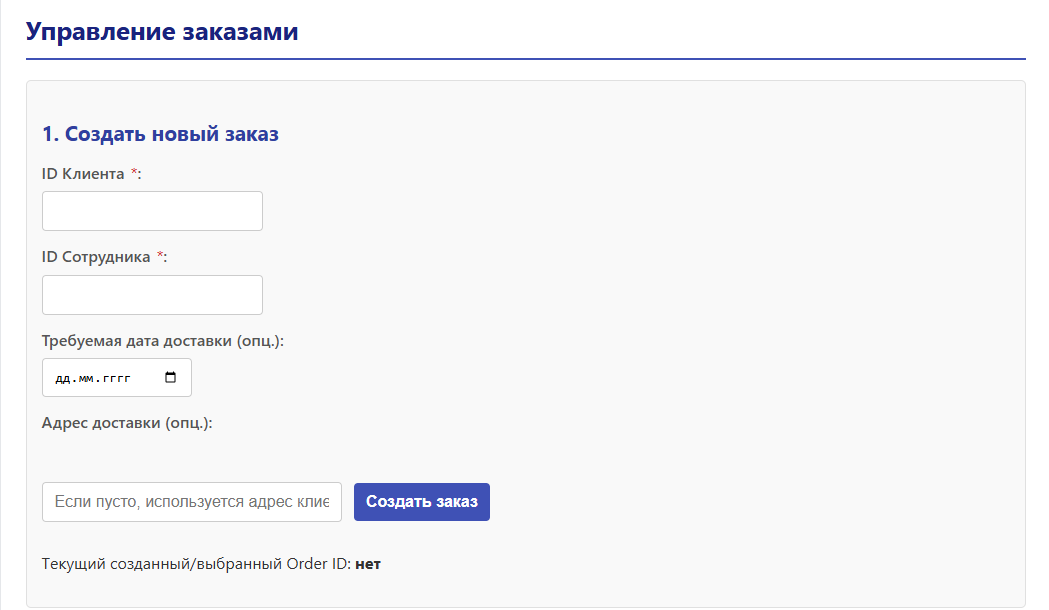


Рисунок 6.2 Интерфейс приложения(2)

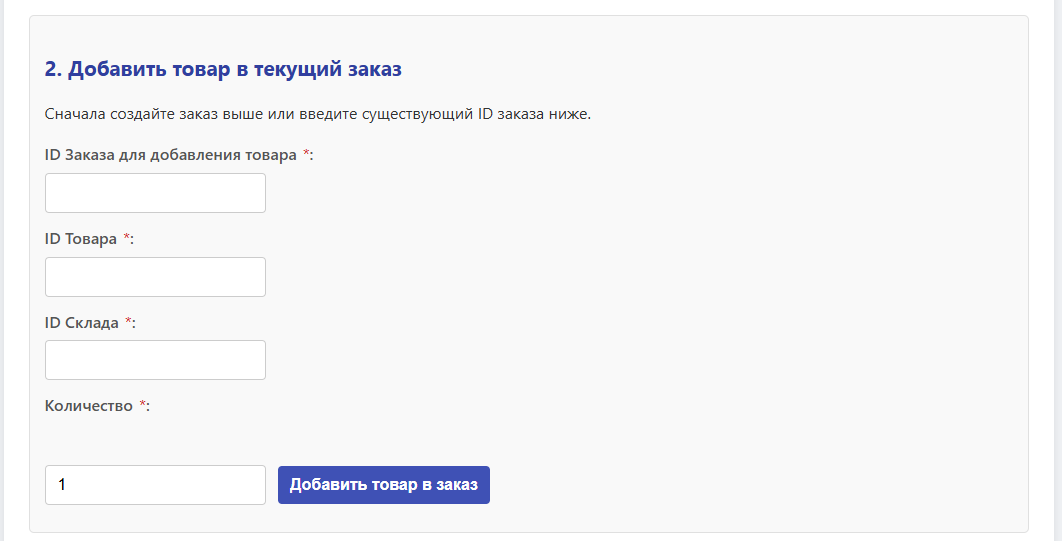


Рисунок 6.3 Интерфейс приложения(3)

**Пример работы приложения:**

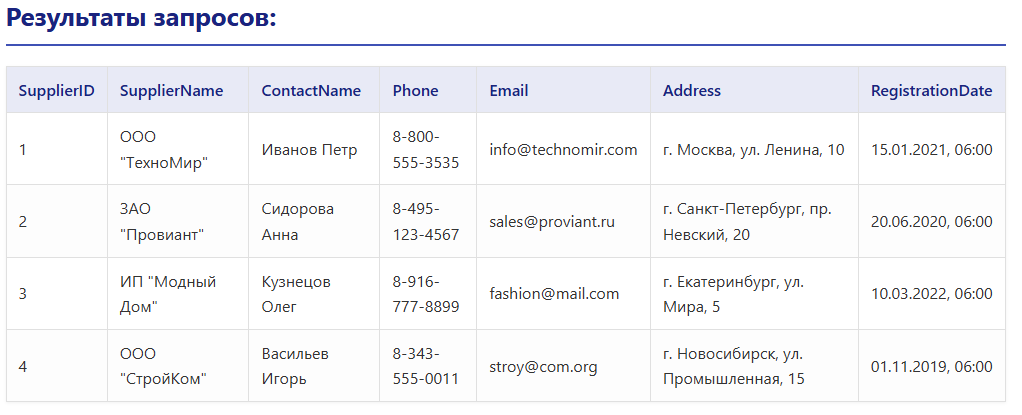
****

Рисунок 6.4 Вызов таблицы Suppliers через интерфейс.



Рисунок 6.5 Вызов view Active Products

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения курсовой работы была успешно решена задача разработки реляционной базы данных «Оптовая база данных». Была проанализирована предметная область, определены основные сущности и их атрибуты, что позволило спроектировать логическую и физическую структуру базы данных.

Реализация базы данных была выполнена средствами СУБД Microsoft SQL Server. Были созданы таблицы для хранения информации о категориях товаров, поставщиках, товарах, складах, остатках на складах, должностях, сотрудниках, клиентах, заказах, деталях заказов, способах оплаты, платежах и для ведения журнала аудита изменений данных сотрудников. Для обеспечения целостности и согласованности данных были использованы первичные и внешние ключи, а также ограничения UNIQUE, CHECK и DEFAULT. Особое внимание было уделено вычисляемому столбцу LineTotal в OrderDetails для автоматического расчета стоимости позиции.

Для упрощения доступа к данным и предоставления информации в удобном виде были разработаны различные представления (VIEWs), как простые, так и сложные, использующие объединения таблиц, агрегатные функции и пользовательские функции.

С целью инкапсуляции бизнес-логики и автоматизации типовых операций были созданы пользовательские функции (скалярные и табличные) и хранимые процедуры. Хранимые процедуры реализуют такие важные операции, как создание заказа, добавление товаров в заказ, обновление статуса заказа, обработка платежей и формирование отчета о товарах с низкими остатками. В ряде процедур применены транзакции для обеспечения атомарности критически важных операций.

Для автоматического поддержания целостности данных и реализации специфической бизнес-логики были разработаны триггеры. Они отвечают за обновление складских остатков при изменении деталей заказа, пересчет общей суммы заказа, предотвращение удаления товаров при определенных условиях и логирование изменений данных сотрудников.

В результате проделанной работы были достигнуты все поставленные цели:

* Проанализирована и формализована предметная область.
* Спроектирована и реализована структура базы данных.
* Освоены и применены на практике механизмы обеспечения целостности данных.
* Разработаны представления, функции, хранимые процедуры и триггеры.
* Получены практические навыки работы с СУБД MS SQL Server и языком T-SQL.

**СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ**

1. Дейт К. Дж. Введение в системы баз данных. – 8-е изд. – М.: Вильямс, 2016. – 1328 с.
2. Коннолли Т., Бегг К. Базы данных. Проектирование, реализация и сопровождение. Теория и практика. – 3-е изд. – М.: Вильямс, 2017. – 1436 с.
3. Гарсиа-Молина Г., Ульман Дж., Уидом Дж. Системы баз данных. Полный курс. – М.: Вильямс, 2018. – 1088 с.
4. Microsoft SQL Server Documentation [Электронный ресурс]. – Режим доступа: [https://docs.microsoft.com/ru-ru/sql/sql-server/](https://www.google.com/url?sa=E&q=https%3A%2F%2Fdocs.microsoft.com%2Fru-ru%2Fsql%2Fsql-server%2F). – Дата доступа: (указать актуальную дату).
5. Ваш SQL-скрипт (Приложение А). (Можно указать как: Собственная разработка SQL-скриптов для СУБД MS SQL Server в рамках курсового проектирования).
6. (Другие источники, если использовались)

**ПРИЛОЖЕНИЕ А**

**Листинг SQL-скрипта создания базы данных WholesaleDB**

[Здесь должен быть вставлен полный текст вашего файла Course.txt]